FINDING HIDDEN RELATIONSHIPS BETWEEN MEDICAL CONCEPTS BY LEVERAGING METAMAP AND TEXT MINING TECHNIQUES

A Paper
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Weikang Yang

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

May 2017

Fargo, North Dakota

North Dakota State University
Graduate School

Title

Finding Hidden Relationships between Medical Concepts by Leveraging MetaMap and Text Mining Techniques

By

Weikang Yang

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Dr. Wei Jin
Chair

Dr. Juan Li

Dr. Na Gong

Approved:

05/26/2017
Date

Dr. Brian M. Slator
Department Chair

ABSTRACT

A lot of efforts have been made in order to make new discoveries in the biomedical filed. However, those valuable information may be hidden in text without applying appropriate text mining techniques. In this paper, I utilize MetaMap, a powerful biomedical tool provided by National Library of Medicine (NLM), along with appropriate text mining techniques, to detect hidden connections between biomedical concepts. The huge volume of Medline documents are used as data source and experimental data, where more than 20 million titles and abstracts of Medline articles are analyzed. On top of this corpus, biomedical concept queries are enabled to allow users to specify any two particular medical concepts, and the system will automatically identify potential relationships that may connect them. A graphical user interface is also developed to facilitate the search process and result presentation.

ACKNOWLEDGEMENTS

It is a great honor for me to get so much support during the completion of this paper, I really appreciate all those peoples' help and kindness. First, I would like to thank my advisor – Dr. Wei Jin for her professionalism and patience to guide me through this study. It will impossible for me to complete this paper smoothly without her help. I would also like to thank committee members – Drs. Juan Li and Na Gong for their help and presence. I would also like to thank the NDSU computer science department for providing the resources I need in this research. For example, Dr. Jeremy Straub provides me a lot of useful information for the journals I should look into in his seminar. I would like to thank CS Administrative Secretaries - Annette Sprague, Jane Dickerson, and Betty Opheim for their patience to help me finish paperwork and remind me of deadlines. At last, I appreciate all the supports from my family and friends which helped me a lot to fulfill my dream in NDSU.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

NLM.....National Library of Medicine

Proc .....Process

MedMeta .....Medline and MetaMap

S2C.....Semantic Type to Concept

C2W .....Concept to Weight

S2CW.....Semantic Type to Concept and Weight

SENT.....Sentence

C2SENT.....Concept to Sentence

TF.....Term Frequency

IDF .....Inverse Document Frequency

CHAPTER 1. INTRODUCTION

Interacting with text is a very common and important activity in people's daily lives. For example, people read books to obtain information and knowledge; people read newspapers to keep up with current events; people use text to communicate with each other by sending emails or messages. According to [1], researchers claim that instead of only acting as passive recipients, people also act as active seekers of texts. In other words, people not only read the text but also try to understand the meaning behind text and then evaluate the importance of texts for them. However, people can still miss some hidden information which is buried in text. For example, we have a sentence indicating "A implies B" in one article and another sentence "B implies C" in another article, and people generally could not link them together to establish the relationship that "A may imply C", especially when they are facing a large volume of data.

Nowadays, with the development of computer science, the process of understanding and analyzing texts are now more relying on the powerful computers, information retrieval techniques, and tools. In 1999, Swanson proposed a system called complementary structures in disjoint literatures" (CSD), which aimed to find hidden knowledge in text. This system implements a simple logic "If concept A influences concept B, and concept B influences concept C, then concept A may influence concept C". This model is usually referred as Swanson's ABC model [2]. For example, in Swanson's study, he examined the relationship between "migraine headache" and "magnesium", and found that although these two concepts do not appear in the same article, they could be logically linked by third party words such as "calcium blockers", "serotonin" etc. [2]

MEDLINE [3] is the "U.S. National Library of Medicine® (NLM) premier bibliographic database that contains more than 23 million references to journal articles in life sciences with a

concentration on biomedicine." This is a perfect data source which can provides necessary medical information for this study. MetaMap is a tool developed by NLM, which can map text to its biomedical interpretation. More details of those tools will be introduced in Chapter 2. In this study, I take the MEDLINE database as data source and process over 23 million articles from it with MetaMap as an aid in mapping articles to their biomedical concepts. On top of it, I build the knowledge discovery model using the technique adapted from Swanson's ABC idea. Given two user-specified biomedical concepts A and B, the system attempts to find the intermediate terms that could possibly connect them, assuming that one or more instances of both concepts occur in the corpus, but not necessarily in the same article. A user interface is also developed that displays the results in a user-friendly way. Comparing with the existing methods, this study makes the following contributions:

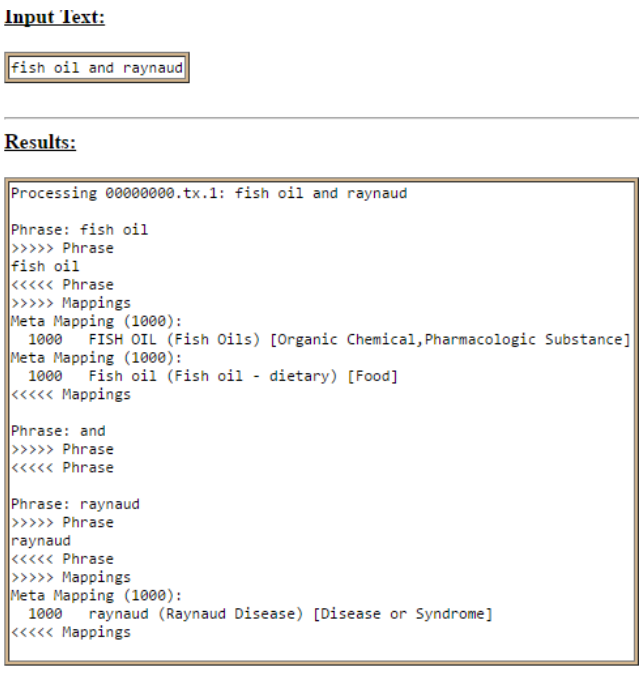
1. Swanson's ABC model has been adapted into the biomedical domain with appropriate domain knowledge incorporated (e.g., semantic type information)
2. A program based on post-processing MetaMap generated output is developed, which can extract from texts to get biomedical concepts along with their associated ontology information.
3. Variant forms of words are considered in this study. A certain biomedical term can have several variants in terms of expression. For example, "Syntaxin" has a preferred name called "Qa-SNARE Proteins" or "Syntaxin Protein". Thanks to the power of MetaMap, it is now possible to treat different variants of the same term under a unique name. This significantly improves the recall of generated result and removes potential duplications caused by variants of words.

4. This study also takes consideration of abbreviations. Abbreviations are very common in biomedical articles, which can be used as disease name, drug name and so on. Existing methods may treat “AIDS” (Acquired Immunodeficiency Syndrome) the same as the verb “aids” if they are not case-sensitive. With the help of MetaMap, the program is also able to distinguish them and produces a better output.

The paper consists of 6 chapters. Chapter 1 will provide a general introduction and objective of this study. Chapter 2 will describe the related studies. Chapter 3 will introduce the methodology used to find hidden knowledge. Chapter 4 shows the User Interface of the Closed Discovery. Chapter 5 evaluates the performance and reports the accuracy of the result. Chapter 6 discusses the future improvement.

CHAPTER 2. BACKGROUND

A lot of efforts have been made to try to map text to its biomedical meaning. For example, researchers around the world have developed tools like MicroMeSH, SAPHIRE and MetaMap to do this job. MetaMap[4] is a public tool developed by the National Library of Medicine (NLM), which can map text to biomedical concepts using its enormous thesaurus. This has been a popular tool applied to many Information Retrieval and text-mining applications [5]. In particular, when MetaMap receives a sentence, it breaks the sentence down into smaller pieces called phrases. Then, MetaMap will normalize those phrases and search its database to find mapping concepts and to calculate a mapping score for each concept. For example, for a short phrase “fish oil and Raynaud”, the human-readable MetaMap output is shown in Figure 2.1.



Input Text:

```
fish oil and raynaud
```

Results:

```
Processing 00000000.tx.1: fish oil and raynaud

Phrase: fish oil
>>>>> Phrase
fish oil
<<<<< Phrase
>>>>> Mappings
Meta Mapping (1000):
  1000   FISH OIL (Fish Oils) [Organic Chemical,Pharmacologic Substance]
Meta Mapping (1000):
  1000   Fish oil (Fish oil - dietary) [Food]
<<<<< Mappings

Phrase: and
>>>>> Phrase
<<<<< Phrase

Phrase: raynaud
>>>>> Phrase
raynaud
<<<<< Phrase
>>>>> Mappings
Meta Mapping (1000):
  1000   raynaud (Raynaud Disease) [Disease or Syndrome]
<<<<< Mappings
```

Figure 2.1. A human-readable output of “fish oil and Raynaud”

```
"Mappings": [
  {
    "MappingScore": "-1000",
    "MappingCandidates": [
      {
        "CandidateScore": "-1000",
        "CandidateCUI": "C0556145",
        "CandidateMatched": "Fish oil",
        "CandidatePreferred": "Fish oil - dietary",
        "MatchedWords": ["fish","oil"],
        "SemTypes": ["food"],
        "MatchMaps": [
          {
            "TextMatchStart": "1",
            "TextMatchEnd": "2",
            "ConcMatchStart": "1",
            "ConcMatchEnd": "2",
            "LexVariation": "0"
          }],
        "IsHead": "yes",
        "IsOverMatch": "no",
        "Sources": ["CHV","MTH","SNOMEDCT_US"],
        "ConceptPIs": [
          {
            "StartPos": "0",
            "Length": "8"
          }],
        "Status": "0",
        "Negated": "0"
      }]
  }]
```


Figure 2.2. JSON output of "fish oil and Raynaud"

A lot of alternative output options are also available for MetaMap. Figure 2.2 above shows part of JSON output for the same phrase "fish oil and Raynaud".

There are already many applications of MetaMap in the field of biomedical studies. For example, Wendy, Marcelo etc. developed a system [6] to detect patients with respiratory illness by processing patients' clinical reports with MetaMap. Zuccon, Holloway etc. try to automatically identify disorders mentioned in health records such as discharge summaries by using MetaMap [7]. In order to test the performance of MetaMap, Pratt and Yetisgen conduct a research [8], which aims to compare MetaMap's capability with that of people who are familiar with the biomedical field. The result shows that MetaMap is capable to identify the biomedical concepts with a 93.3% of recall comparing with those tagged by biomedical experts.

Knowledge discovery in biomedical texts originates from Swanson's ABC model [2], based on which Jin and Srihari proposed a new type of query, namely, concept chain queries, attempting to detect hidden links between concepts. In [9], they tried to generate concept chains connecting topics of interest from counterterrorism documents. The approach used a variant of Term Frequency (TF) - Inverse Document Frequency (IDF) weighting scheme for evaluating each potential connecting term, achieving an approximate 82.5% recall. This work is different from Jin's study by using a completely different context – biomedical domain instead of counterterrorism corpus. Another related work was proposed by Gopalakrishnan, Kishlay, etc. [10], which introduced a new approach that created a graph-based knowledge base and then used a bi-directional search for finding paths in the graph to answer concept chain queries in the biomedical field. My method has been compared with their work and the comparative result is presented in the experimental section.

CHAPTER 3. THE PROPOSED METHOD

3.1. Overall System Design

This system applies the principle of pipe and filter software architectural design [11] which has many modules in the system acting as filters. Each filter will take output from the previous filter and produce intermediate data as input to the next filter. In this design, the system creates a data flow where the original data flows through each filter and gets modified by them, and eventually produces the desired output at the end of the flow. Figure 3.1 shows the overall architecture design. The system contains three main phases –MetaMap processing phase, preparation phase, and output phase. In MetaMap processing phase, the system has MedMeta module which takes Medline documents as input and extracts useful information from them. Then, MedMeta module will generate MetaMaped documents in XML format as output of this phase. Section 3.2 will introduce the structure of Medline documents and what information has been extracted by the MedMeta module. Section 3.3 will explain the design of MedMeta module in detail. In preparation phase, the system has two modules – S2C generation module and title & abstract fetching module. They take MetaMaped documents as input and generate S2C documents. Section 3.4 and section 3.5 will reveal the design and function of S2C generation module and title & abstract fetching module. The last phase is the output phase. In this phase, the system has the closed discovery module which applied the adapted Swanson's ABC model to find hidden links between input concepts and build concept chains. This module also contains a GUI (graphical user interface) that shows generated concept chains and evidence related to each chain. Section 3.6 will talk about the design and features of ClosedDiscovery module with some screenshots of the GUI. The outputs of closed discovery module are S2CW documents and

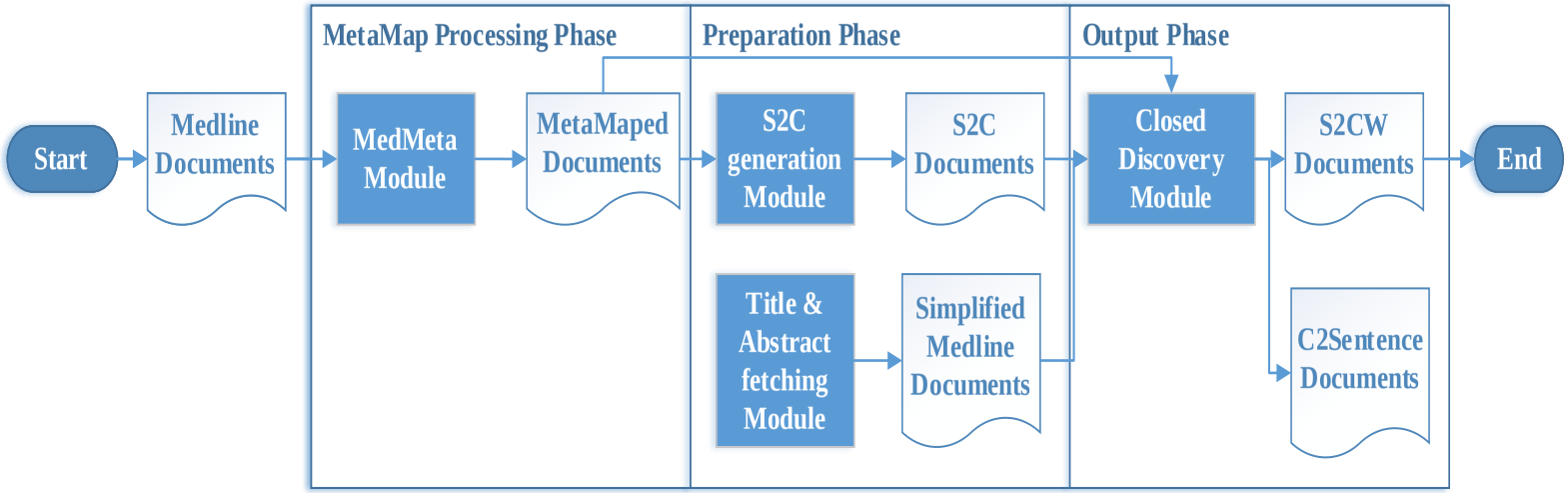


Figure 3.1. Overall system design

C2Sentence documents and they are the final outputs of this pipe and filter system. The details of those documents will also be covered in section 3.6.

3.2. Medline Documents

Medline documents are released by NLM and are used as the data sources for the system. According to the description on the NLM website [3], Medline documents contain the basic information such as abstract and title of a biomedical paper submitted to Medline worldwide. “23 million references to journal articles in life sciences with a concentration on biomedicine” have been collected so far [3]. Medline provides free access to Medline documents and those documents can be downloaded by the following link: ftp://ftp.ncbi.nlm.nih.gov/pubmed/baseline. In this study, I downloaded 746 Medline documents from the link and each Medline document contains 30,000 Medline citations. Each Medline citation is a reference to an article so 22,380,000 article references are processed by the system. Figure 3.2 illustrates an example of Medline citation. Notice that since each Medline citation has more than 100 lines, many of XML tags have been folded so that the figure can show the whole picture of it. Highlighted parts in the figure 3.2 are the parts useful to the system and they are gathered and processed by the system. The XML tags of those useful information are <PMID>, <PubDate>, <ArticleTitle>, and <AbstractText>. <PMID> tag contains the information of an 8-digit PubMed unique identifier; <PubDate> tag contains the publishing date of the article; <ArticleTitle> tag contains the entire title of the journal article and it is always written in English and non-English title will be translated first; <AbstractText> tag contains the English-language abstract of the article.

Figure 3.2. An example of Medline Citation

3.3. MedMeta Module

MedMeta module is the first module in the system. The function of this module can be described by the following four sequential steps: (i) extracts useful information mentioned in Section 3.2 from data source – Medline documents; (ii) inserts MetaMap API in the code and sends titles and abstracts to the MetaMap Server; (iii) reads from MetaMap output and builds indexes of Medline documents based on concept occurrence relationship. (iv) Write index files in XML format. Figure 3.3 is a UML Sequence Diagram that explains how objects in MedMeta interact with each other.

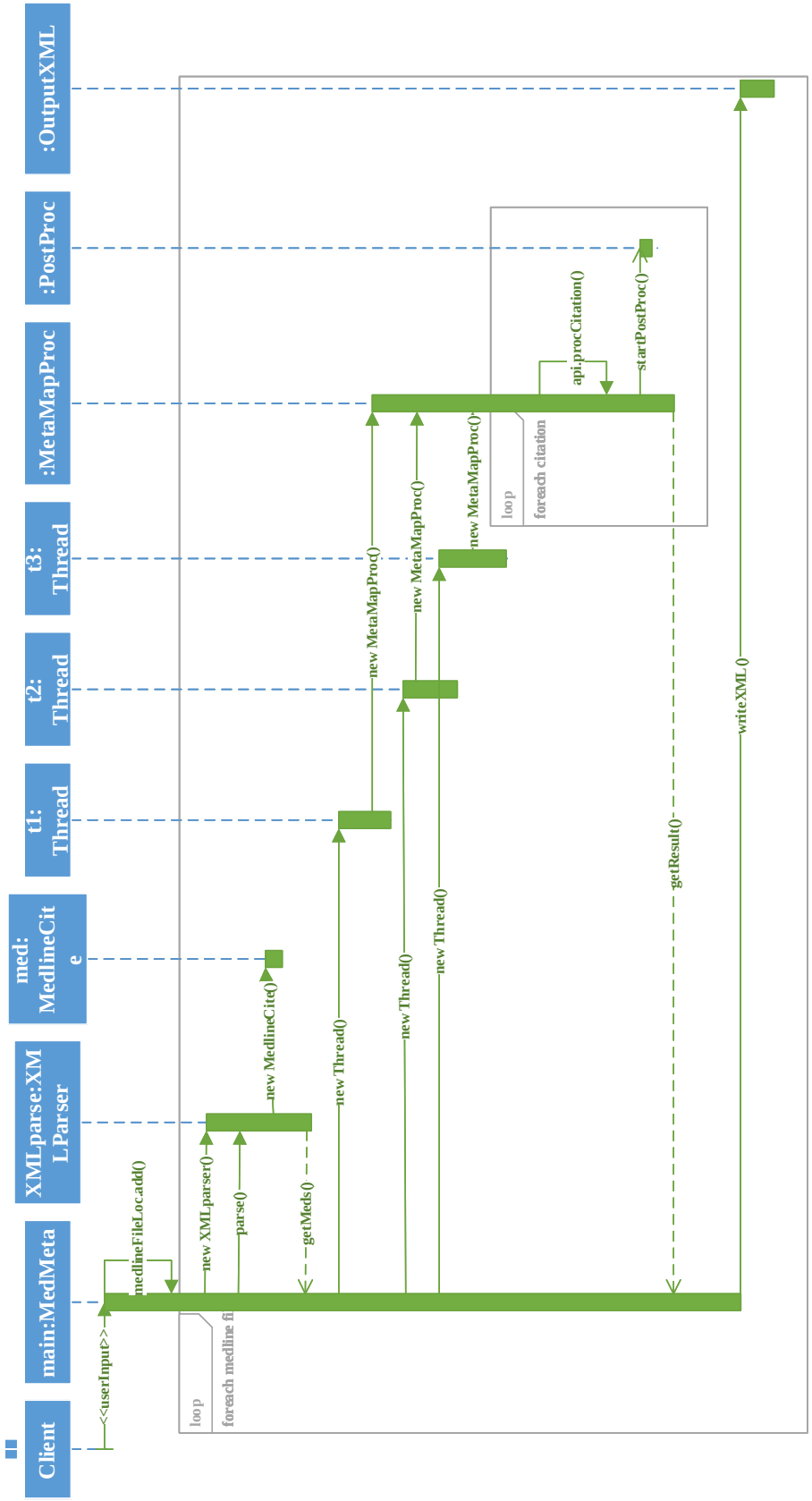


Figure 3.3. Sequence diagram for MedMeta Module

Figure 3.3 shows that the MedMeta Module consists of seven classes – MedMeta, XMLParser, MedlineCite, MetaMapProc, PostProc, OutputXML, and Thread. The following part discusses the details of each class.

- **MedMeta**: MedMeta class contains the main function, so the main thread begins in the object of this class. It is the class that regulates the creation of other objects and invokes methods from other objects.
- **XMLParser**: XMLParser class uses XMLEventReader which is a popular XML parsing Java API to parse required data. Instead of reading the entire XML structure into memory and parsing it, XMLEventReader iterates over XML file and is triggered if and only if a certain event happens. As a result, XMLEventReader is effective for large XML files which might consume a huge amount of memory by using the traditional way. In this study, the total size of Medline documents is 93.7 GB with single document as large as 200MB, so XMLEventReader is a better tool for parsing Medline documents. Appendix A shows the code I used in parsing those Medline documents. After paring the Medline documents, XMLParser creates MedlineCite objects and stores data into those objects.
- **MedlineCite:** It is a placeholder class that stores the data parsed from Medline documents. It has four data fields – article title, article abstract, PMID, and publishing date.
- **MetaMapProc**: The most important class in MedMeta Module. This class handle the task of communicate with MetaMap Server and create PostProc object to build the index. Figure 3.4 shows a code snippet from MetaMapProc where communication with MetaMap Server happens. Notice that the setOption() method will set the input option for MetaMap server. The "-y -K -Q 0" option I used here means the server will use Word

Sense Disambiguation server and ignore stop phrases. Titles and abstracts are sent to MetaMap Server by calling processCitationsFromString(content) method. Figure 3.5 shows the MetaMap server is running and communicating with the system. After processing the string, MetaMap Server will return the result to MetaMapProc object and MetaMapProc objects will send result to PostProc object and start post processing.

```
// Start a connection to submit the data, the default server is
// localhost
MetaMapApi api = new MetaMapApiImpl(host, port);
// Metamap behavior options
api.setOptions("-y -K -Q 0");

if (title != null) {
    //send title to MetaMap Server
    List<Result> tResults = api.processCitationsFromString(title);
    if (tResults.size() > 0) {
        tResult = tResults.get(0);
        p1 = new PostProcessor(tResult, PMID, pubDate, "T", semLst);
        p1.startPostProcess();
    }
}
if (articleAbstract != null) {
    //send abstract to MetaMap Server
    List<Result> aResults = api.processCitationsFromString(articleAbstract);
    if (aResults.size() > 0) {
        aResult = aResults.get(0);
        p2 = new PostProcessor(aResult, PMID, pubDate, "A", semLst);
        p2.startPostProcess();
    }
}

// disconnect the apiImplement to avoid open_too_many_files error
api.disconnect();
```

Figure 3.4. A code snippet from MetaMapProc class.

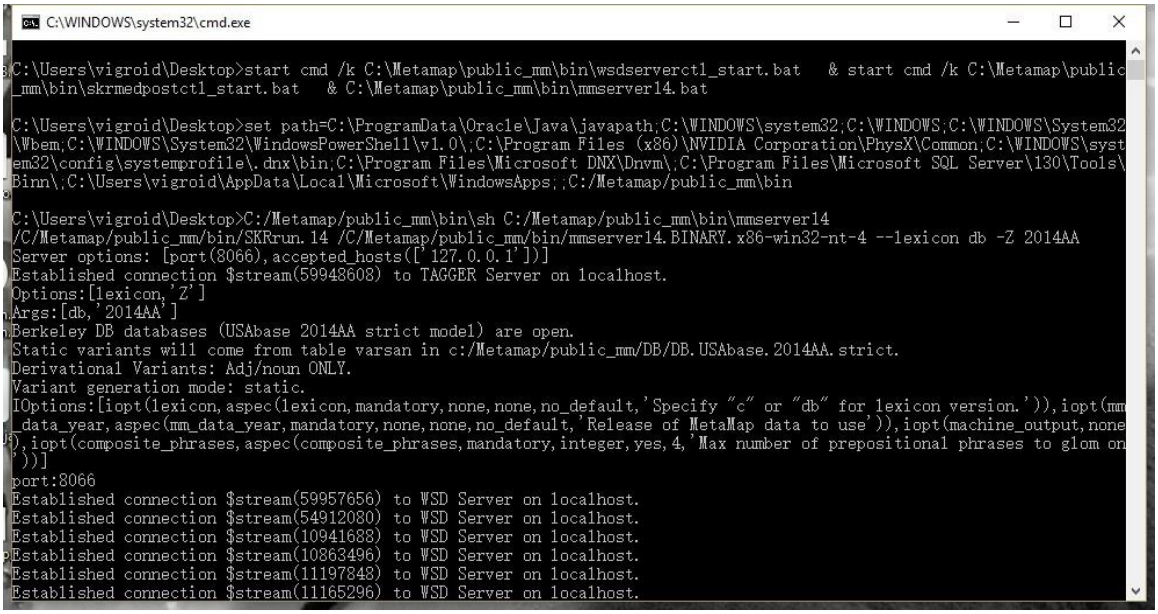


Figure 3.5. A running MetaMap Server.

- **PostProc:** Object of this class will take results from MetaMap server as input and gather information from results. Then, it creates a three-level data structure that holds the data gathered from result. Figure 3.6 illustrates the design of this three-level data structure. The first level is **Semantic Type** objects, it is obtained by calling getSemanticTypes() method and it holds one or many Concepts objects. The second level is **Concept objects**, it is obtained by calling getConceptName() method and it holds one or many Occurrence objects. The bottom level is **Occurrence objects**, it represents a certain occurrence of a word in the Medline documents. The Occurrence objects hold the information such as OccurrenceName(the original word appears in the text), PreferName(the prefer name of the original word), Tag(location where the word appears in the Medline citation. “T” stands for title; “A” stands for abstract), PMID (PMID of the Medline citation where the word appears), “PubYear” (publish

date of the Medline citation where the word appears), and Offset("the exact location of the word appears in the title or abstract", [(23, 8)] in this case means that the word starts from 23th character of the string and occupies 8 characters in total).

```
<?xml version="1.0" encoding="UTF-8"?>
<Sementic name="aapp" description="Amino Acid, Peptide, or Protein">
   <Concept>
      <ConceptName>Oxidases</ConceptName>
      <TitleCount>1</TitleCount>
      <AbstractCount>0</AbstractCount>
      <Occurence>
         <OccurenceName>oxidases</OccurenceName>
         <PrefName>Oxidase</PrefName>
         <Tag>T</Tag>
         <PMID>16748552</PMID>
         <PubYear>1949</PubYear>
         <Offset>[(23, 8)]</Offset>
      </Occurence>
   </Concept>
   ...
</Sementic>
```

Figure 3.6. A piece of MetaMaped Document.

- **OutputXML:** The object of this class will take the three-level data structure as input and create XML files to hold the data. Figure 3.6 shows an example of MetaMaped Document created by OutputXML object. In this study, OutputXML object generated 161GB of MetaMaped Documents in this study overall.
- **Thread:** The object of this class will create a new thread which has a MetaMapProc object that implements Runnable Java Interface. As we see from the description of OutputXML class, the MedMeta module read, processed and generated huge amount of data. To improve the performance of MedMeta Module, it is wise to apply parallel processing methodology such as multithreading into the process of developing this module. MedMeta Module first divide 30,000 Medline citations in each Medline document into 3 parts so each part has 10,000 Medline citation. Then, MedMeta

Module creates a new thread for each part of Medline citations so there will be three threads running at the same time. Each thread will communicate with its own MetaMap server and create its own PostProc object for the postprocessing. After the execution of each thread, three threads will join and hand the control of CPU back to main thread which starts OutputXML process later. As a result, the MedMeta Module can communicate with three MetaMap servers which run at different ports on the same machine at the same time. In the Evaluation section of this paper, I will show how the performance has been improved by applying multithreading into the system comparing with single thread. Figure 3.7 shows the user interface of MedMeta module. Notice that the user interface displays both thread number and file number for each MetaMap execution.

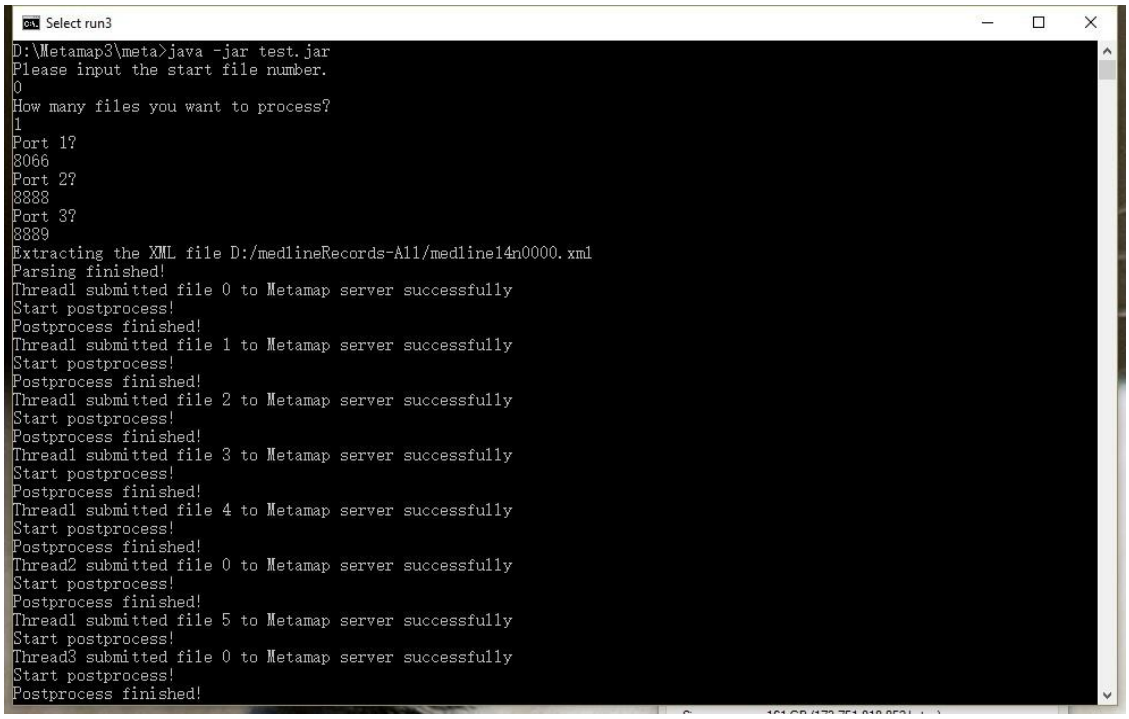


Figure 3.7. User Interface of the MedMeta Module

3.4. S2C Generation Module

S2C generation module is one of the two modules in the preparation phase. The main purpose of this module is to generate a simplified version of MetaMaped documents which called S2C documents. As mentioned in section 3.3, the MetaMaped XML documents has a three-level data structure – Semantic Type, Concept and Occurrence. However, in latter part of the system, a two-level relationship – Semantic Type and Concept (i.e. S2C) is used frequently. If the system gets the two-level relationship from the three-level MetaMaped documents every time the relationship is requested, it will take a huge amount of time to iterate over all MetaMaped documents several times. By reducing three-level relationship down to the S2C two-level relationship, it also reduces the access time. After running S2C generation module, the 161GB of three-level MetaMaped documents are simplified to 42MB of two-level S2C documents. Figure 3.8 shows the design of S2C generation module.

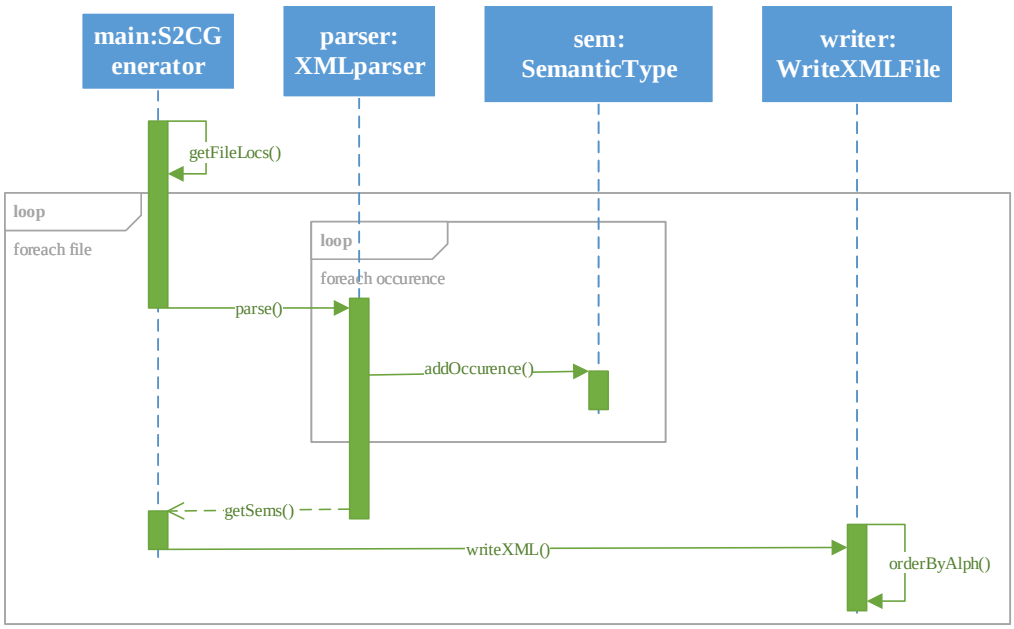


Figure 3.8. Sequence diagram for S2C generation module

S2C generation module consist of three classes – S2CGenerator, XMLParser, SemanticType and WriteXMLFile. The following part will discuss the detail of each class.

- **S2CGenerator:** This class contains the main method. S2CGenerator controls the creation of other objects and invoking of their method. The first step of this class is to get the file path of MetaMaped documents. After finding the path of MetaMaped documents, it creates XMLParser and calls the parse() method to build the S2C two-level relationship. Then, it creates WriteXMLFile object and calls the writeXML() method to write the two-level relationship into XML files.
- **XMLParser:** The design of XMLParser in S2C generation module is very similar to the design of XMLParser in MedMeta module. They used the same XML parsing API – XMLEventReader which is event-driving based parsing technique and the only difference is that triggering events are different. XMLParser will iterate over all concepts in MetaMaped documents and add them to SemanticType by calling the addConcpet() method from SemanticType class.
- **SemanticType:** The object of this class holds the S2C relationships by having a Hash set of concepts. When the addConcpet() method is called to add concept into the hash set, it will first check if the hash value exists in the hash table and will only add the concept into the hash set if it has a unique hash value. There are 133 semantic types in total and each of them hold a hash set of concepts.
- **WriteXMLFile:** This class write the S2C relationships stored in SemanticType objects to XML files. Unlike the OutputXML class in MedMeta module, WriteXMLFile uses DocumentBuilderFactory Java API to build the XML files. This avoids errors occurred in

OutputXML where the object may write XML reserved characters such as “&” and “<” into the XML files. The contents of S2C documents is illustrated in Figure 3.9.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SemanticType name="aapp">
    <Concept>'Malic' enzyme</Concept>
    <Concept>(+)6a-hydroxymaackiain 3-O-methyltransferase</Concept>
    <Concept>(-)-Menthol dehydrogenase</Concept>
    <Concept>(123)I-mAb 14C5</Concept>
    <Concept>(125)I-L-PC</Concept>
    <Concept>(125)I-YLFQPQRFamide</Concept>
```

Figure 3.9. A piece of S2C document

3.5. Title & Abstract Fetching Module

The main aim of this module is to fetch title and abstract from Medline documents and builds simplified Medline documents to store title and abstract. As we know, Medline documents are huge and accessing them takes a lot time. If the system only needs the data from title and abstract data field, there is no point to iterate over all data fields in Medline documents to get them. Figure 3.10 shows the design of title & abstract fetching module.

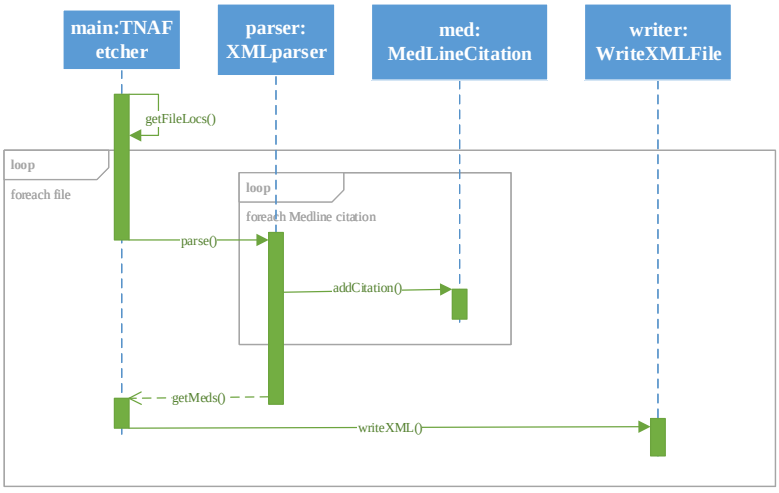


Figure 3.10. Sequence diagram for Title & Abstract Fetching Module

Title & abstract fetching module consist of four classes such as TNAFetcher, XMLParser, MedLineCitation and WriteXMLFile. Since the design of TNAFetcher, XMLParser, and WriteXMLFile is like their counterpart in S2C generation module, they will not be introduced again.

- **MedLineCitation:** This class has two data fields – title and abstract. Those two data fields are used to store data fetched from XML documents. Figure 3.11 shows a piece of simplified Medline document generate by MedLineCitation object.

```
<MedLineCitation PMID="2447742">
    <Title>Latex agglutination test for alpha-fetoprotein in the diagnosis of premature
    rupture of the amniotic membranes (PROM).</Title>
    <Abstract>A rapid latex agglutination test for alpha-fetoprotein (AFP) was compared with
    a pH-indicator and patient history in the detection of premature rupture of the amniotic
    membranes. Of 120 patients examined, 34 had an established rupture of the membranes, and
    56 had suspected rupture. Thirty patients had no evidence of membrane rupture. The
    vaginal content was examined with a pH-indicator. Samples of vaginal content were also
    obtained to perform the latex agglutination test, and 103 of these samples were analysed
    by a radio-immunoassay (RIA) technique for AFP. Our results indicate that the latex
    agglutination test is of doubtful value due to the many inconclusive test results. The
    sensitivity of the latex test is less than 15%, and the specificity is 80%. Patient
    history combined with pH-indicator test is far more informative than the latex
    agglutination test.</Abstract>
</MedLineCitation>
```

Figure 3.11. A piece of simplified Medline Document

3.6. ClosedDiscovery Module

The ClosedDiscovery Module takes user inputs, MetaMaped documents, S2C documents, and simplified Medline documents as inputs. After processing, the module generates S2CW files and display its finding in a graphical user interface. Figure 3.12 shows the design of ClosedDiscovery Module.

The ClosedDiscovery Module consists of eight classes – InitialPage, ConfigMeta, S2CW, MetaMapProc, ResultProc, WriteXML, SentPage, and MergedSentPage. SentPage and

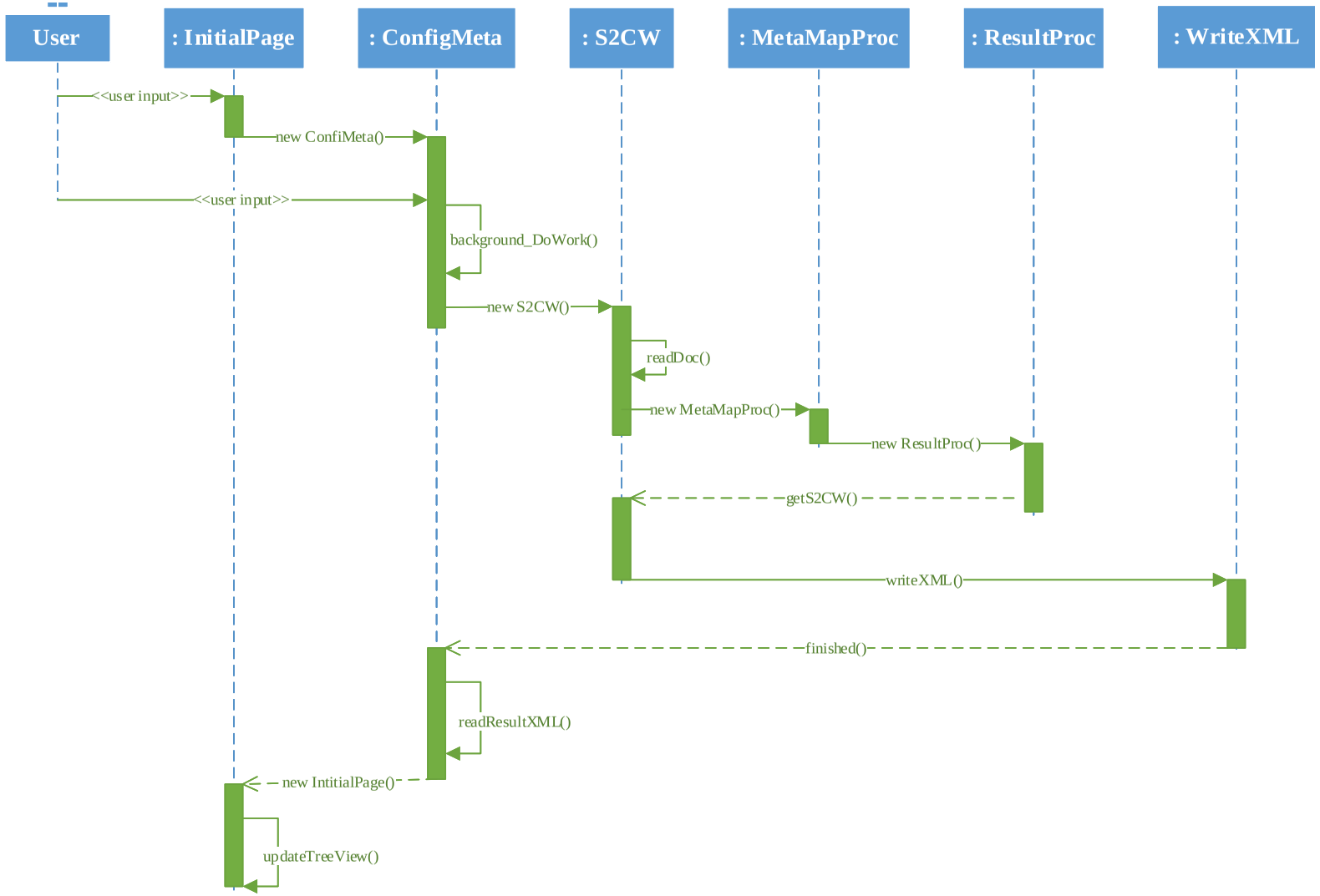


Figure 3.12. Sequence diagram of Closed Discovery module

MergedSentPage are not included in Figure 3.13, but they are used to display additional graphical user interface.

- **InitialPage:** It is a Windows Form object which displays the initial graphical user interface. Figure 3.13 illustrates an example of InitialPage. A InitialPage object has two textboxes to receive inputs from users, and those inputs will be called Topic A and Topic C. Since there are no findings for concept chains yet, the three tree views are empty. Figure 3.xx shows how it looks like if concept chains are generated (i.e. S2CW and C2Sent documents are generated). After clicking the Start button, A ConfigMeta object will be created.
- **ConfigMeta:** It is a Windows Form object that interact with user to get S2C documents. Figure 3.14 shows an example of ConfigMeta object. After clicking the Load Mapping button, user can choose S2C documents they are interested and unselected documents will be ignored. After clicking the Start button in ConfigMeta object, the program will start to find sentences related to Topics user inputted on InitialPage. Then, ConfigMeta object builds XML documents docListA.xml and docListC.xml to store sentences related to Topic A and Topic C. Appendix B shows the structure of docListA.xml and docListC.xml. If those XML files are written successfully, ConfigMeta executes a Java program – S2CWGenerator.
- **S2CW:** The main method of the S2CWGenerator locates in the S2CW class. It reads docListA.xml and docListC.xml which are generated in ConfigMeta object. Then, it creates MetaMapProc objects and passes sentences read from docList documents to the MetaMapProc objects.

- **MetaMapProc:** MetaMapProc class is used to interact with MetaMap Server. It receives sentences from S2CW object and creates MetaMap Java API. Alike the MetaMap Java API used in MedMeta module, the API used in here also requires server address and port number. After sending the sentences to the MetaMap Server, the server will return results back to the MetaMapProc object as Result objects. MetaMapProc object then creates ResultProc to deal with Result objects.
- **ResultProc:** This class is one the most important classes in the CloseDiscovery Module. ResultProc takes MetaMap Result objects as input and builds concept chains by calculating TF (Term Frequency) and IDF (Inverse Document Frequency) [13] in sentences relevant to the input topics. The TF of a concept is the number of times the concept appears in all sentences examined with multiple appearances of a concept in the same sentence counted as one appearance. The IDF measures the importance of the concept and is measured for concept *c* by $IDF(c) = log_e$ *(Total number of sentences / Number of sentences with concept c in it)*. The assumption is that the rarer a concept appears in the sentences, the higher the IDF value is, meaning the more important the concept is to the context. Then, the ResultProc multiplies TF and IDF to get concept weights (i.e. $Weight_c = TF_c * IDF_c$). As we know, each MetaMap Result may contain several concepts. By iterating over all the Results received from the MetaMap Server, the ResultProc can build C2W(Concept to weight) relationship and C2Sents(Concept to Sentencs) relationship. The next step is to join the S2C(Semantic to Concept) relationship and the C2W(Concept to weight) relationship together to get the S2CW relationship,

based on which the weight of each concept is further normalized by:

$$NormalizedWeight(c) = \frac{the\ weight\ of\ the\ concept\ c}{maximum\ weight\ in\ this\ Semantic\ Type}$$

After normalization, ResultProc will sort the weights of concepts in each semantic type in the descending order. The next step is to merge S2CWA(S2CW for Topic A) and S2CWC(S2CW for Topic C) to get the intermediate level information S2CWB. S2CWB represents the potential linking terms between topics A and C and is generated by finding the intersection of S2CWA and S2CWC. In other words, every concept in S2CWB should appear in both S2CWA and S2CWC profiles. The weight of concept in S2CWB is calculated by adding the concept weights in S2CWA and S2CWC. Appendix C shows the structure of S2CW files and Appendix D shows the structure of C2Sent files.

- **WriteXML:** This class will build XML files for S2CWA, S2CWB, S2CWC, C2SentsA, and C2SentsC. It uses DocumentBuilderFactory Java API to build the XML files.
- **SentPage:** This is a Windows Form class and it will be displayed if the user double clicks the node of tree views in the InitialPage. This class will list out all sentences related to a certain concept. Figure 4.7 shows the screenshot of this windows form.
- **MergedSentPage:** This is a Windows Form class and it will be displayed if user double clicks the node of merged tree views in InitialPage. This class will list out all sentences related to concept chains A-B and B-C. Figure 4.8 shows the screenshot of this windows form.

CHAPTER 4. GUI OF CLOSEDDISCOVERY

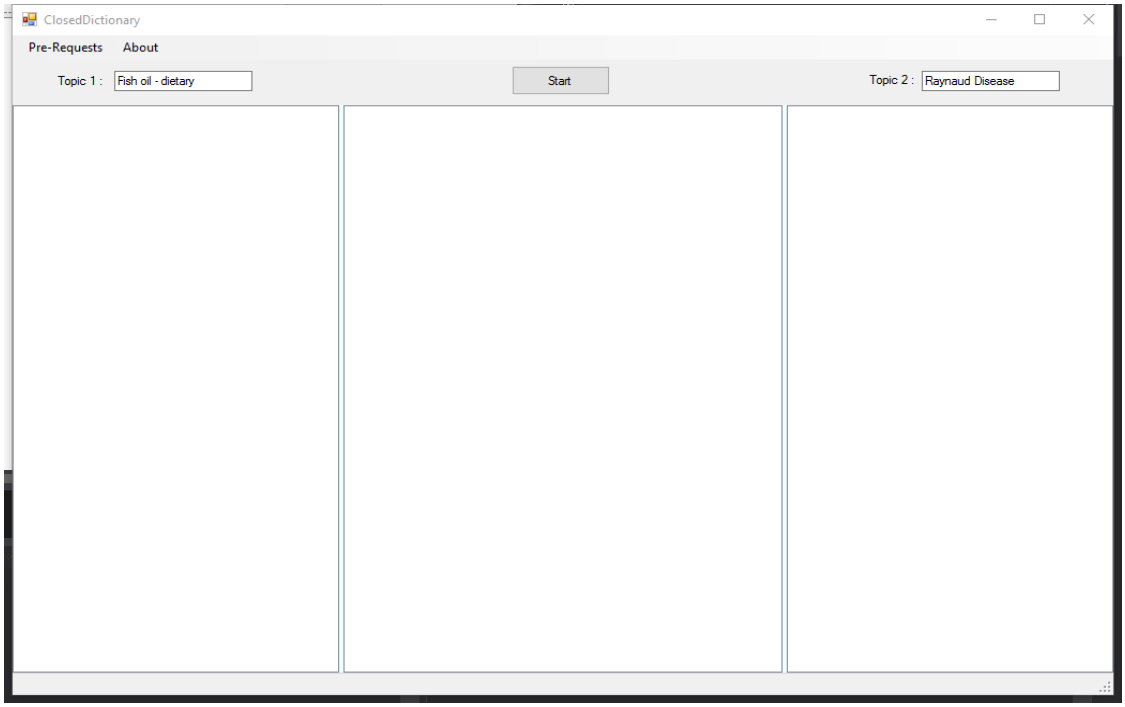


Figure 4.1. The initial page. User types topic A, C and click start button.

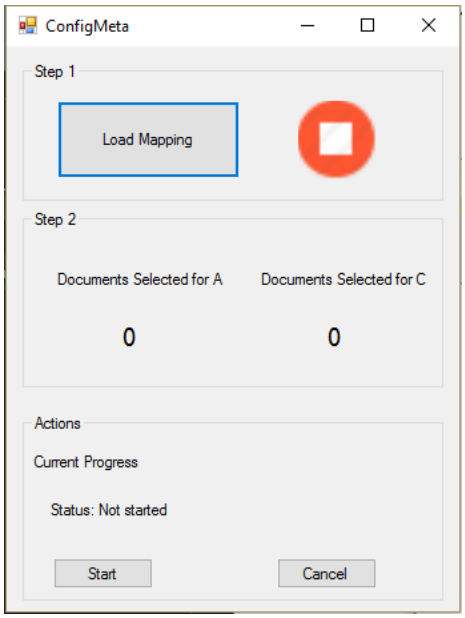


Figure 4.2. ConfigMeta Page appears. User clicks Load Mapping button.

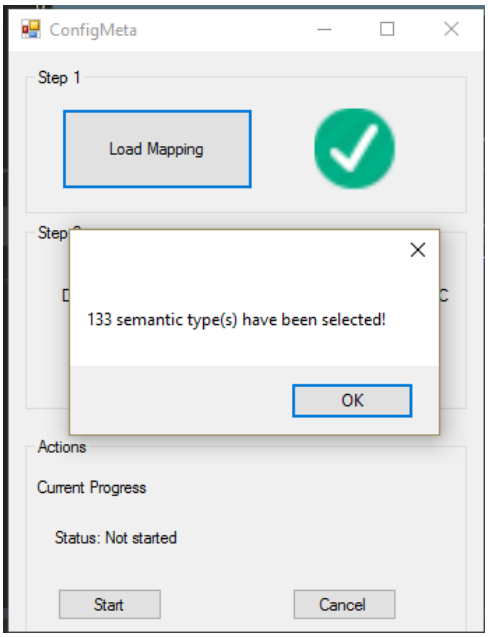


Figure 4.3. Select S2C files and click ok. A Message box appears and shows how many semantic types are selected and load icon is changed.

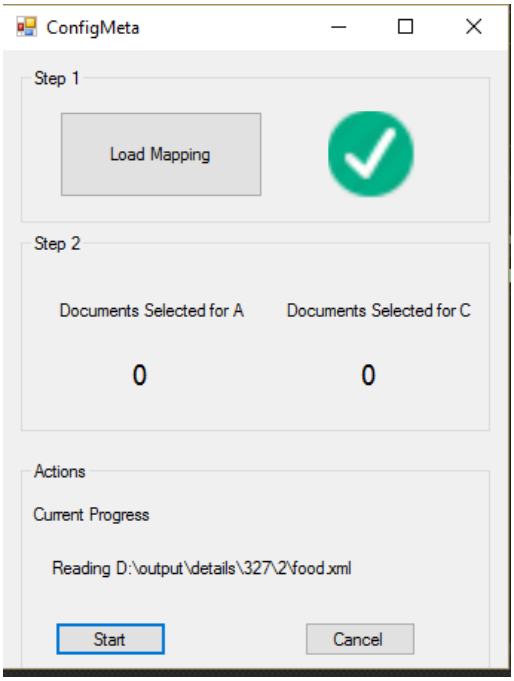


Figure 4.4. Clicks the Start button. The process of finding sentences related to topics of interest starts, and Current Process field shows which file is been processed now.

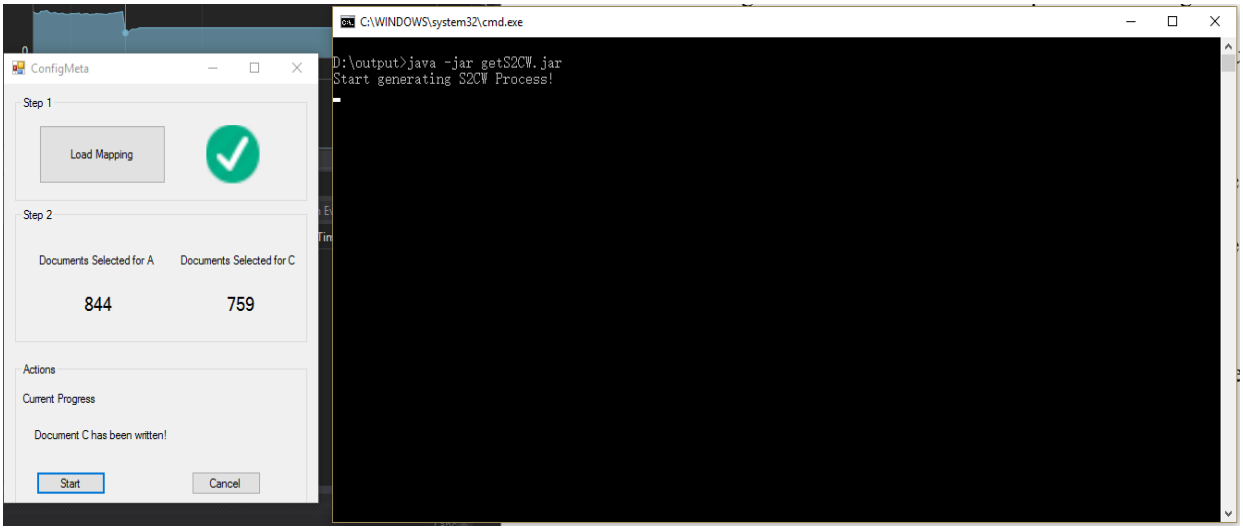


Figure 4.5. After finding of sentences completes, the java program S2CW generator is executed to generate S2CW and C2Sent files.

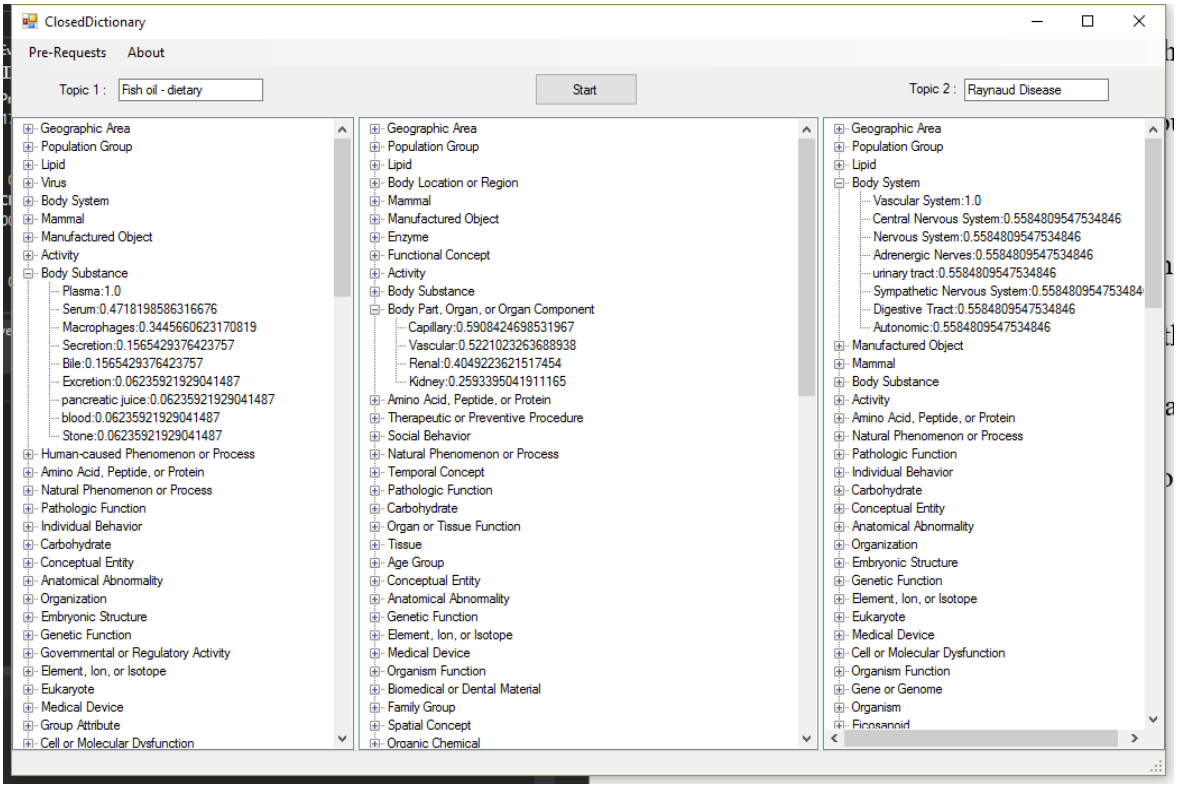


Figure 4.6. InitialPage loads S2CW and C2Sent files into tree views. Left tree view is profile for topic A. Right tree view is profile for topic C. Middle tree view is the merged profile file for both A and C, corresponding to linking concepts between A and C.

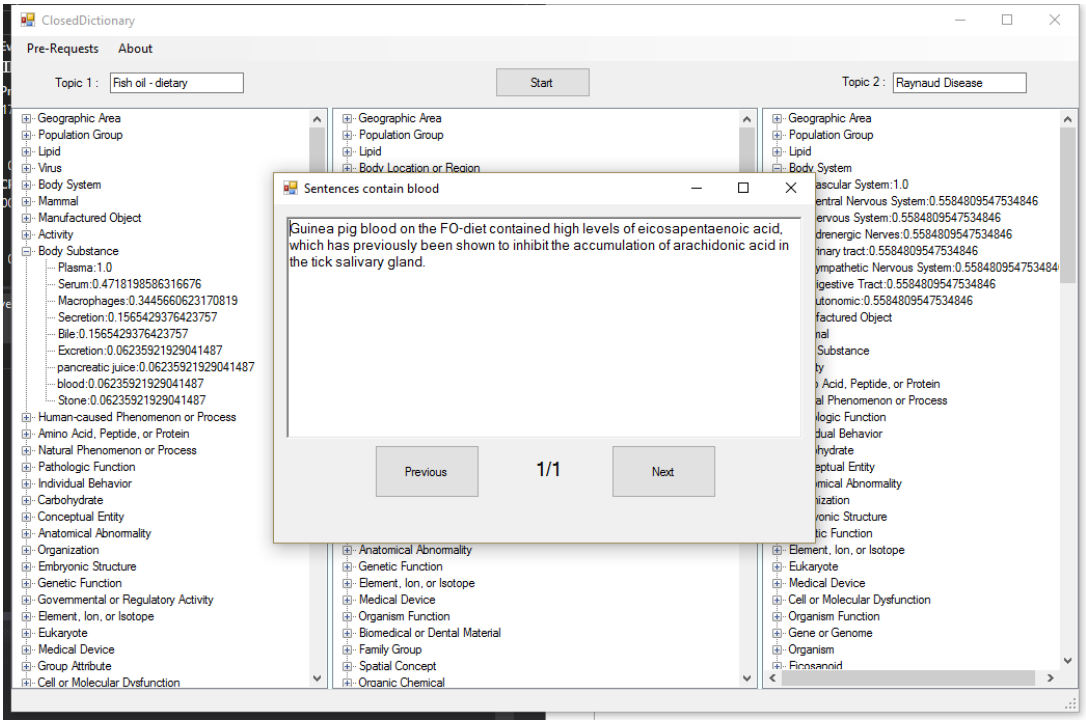


Figure 4.7. When double click a concept node in left or right tree view, a pop-up window appears with all sentences that have this concept.

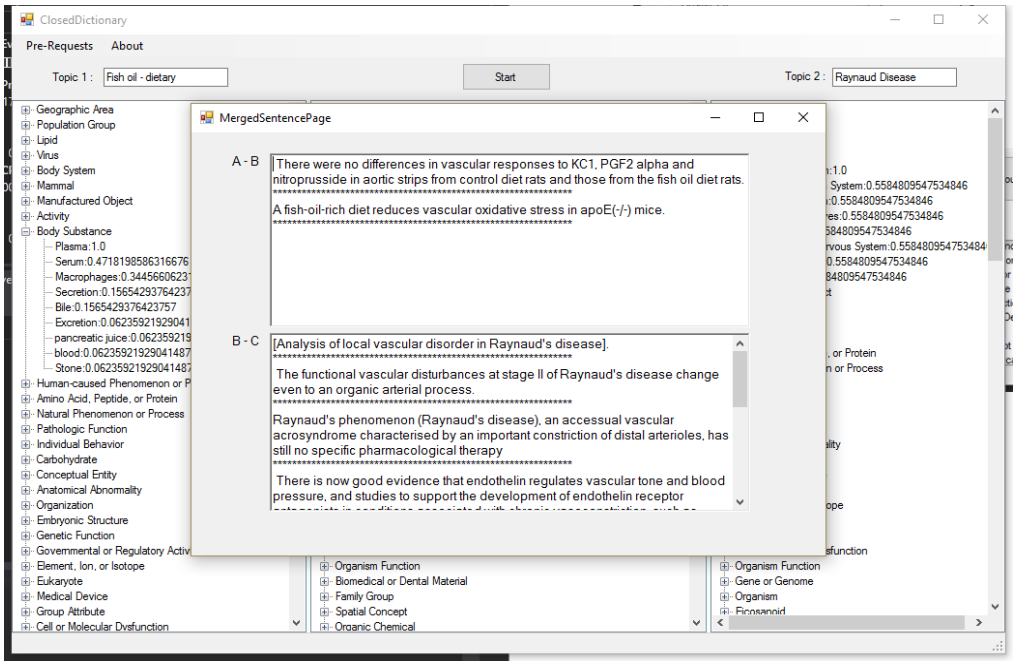


Figure 4.8. When double click a concept node in middle tree view. MergedSentPage window appears with all sentences related to the intermediate linking concept.

CHAPTER 5. EVALUATION & RESULT

5.1. Single Thread vs. Multithread Performance

A lot of changes have been made in the development of this system, and most of them happen in the development of MedMeta Module because it is at the early stage of development of the overall system when everything is not clear yet. The Major Changes I made in the development of MedMeta Module is that I changed the program from a single-thread program into a multi-thread platform. The changed program has three threads used to communicate with the MetaMap server. To show the performance change after applying multithreading, I designed an experiment to test how the performance changes if the number of threads used to communicate with the MetaMap server changes. The machine used in the experiment has an eight-core i7-4790k CPU and 16GB of memory running at Windows Pro 10 x64. The testing Medline document contains 180 Medline citations. The number of threads varies from one to three.

Table 5.1. Performance comparison

Number of thread	Processing Time	Average Memory Usage	Average CPU Usage	Performance compare to single thread	% of theoretical performance
1	145 s	98MB	25%	1	
2	87 s	110MB	40%	1.67	85%
3	62 s	128MB	73%	2.34	78%

As we can see in Table 1, the performance improves significantly as the number of threads increases. When there are two threads, the processing time decreased from 145 seconds to 87 seconds so the performance increases 1.67 times. When there are three threads, the processing time decreases from 145 seconds to 62 seconds so the performance increases 2.34

times. We can also see that the increasing speed slows down when there are more threads involved. When we have two threads, the actual performance 1.67 is 83.5% of the theoretical best performance of 2. When we have three threads, the actual performance of 2.34 is only 78% of the theoretically best performance of 3. Besides, running many MetaMap servers costs a huge amount of memory since each of them could consume 3GB of memory. Since I only have 16 GB of memory, I choose three threads as the best number of threads to run in this study because it keeps a good balance between performance and memory usage.

5.2. Result Evaluation

The system developed has found the meaningful connecting terms between two input topics A and C, and stored those information in the S2CWB XML files. To test if the output of the system is accurate, we need to compare our results with the findings from other studies. In Gopalakrishnan and Kishlay's study [10], they have run their own algorithm on Mesh terms related discovery between input topics and found connecting Mesh terms that have been proved to be meaningful in explaining the relationship between input topics. In this section, I attempt to run the same query pairs with the same data collection but not in a Mesh term context (instead I use the real citation title and abstracts as the corpus). Then, I compare my results with Gopalakrishnan and Kishlay's results to see if my results are consistent with theirs or identify some new findings. The input query pair "Fish-oil and Raynaud's Disease" will be used as a testing case, i.e., topic A is fish-oil and topic C is Raynaud's Disease.

5.2.1. Fish Oil – Raynaud's Disease

Gopalakrishnan and Kishlay had found connecting words such as platelet aggregation, vascular reactivity, blood viscosity, Prostaglandin, and arthritis rheumatoid [10]. Table 2 shows how our system behaves for finding those meaningful connections.

Table 5.2. Fish oil and Raynaud Disease

Connecting Concept	Find?	Weight	Rank in the Semantic type
Platelet aggregation	True	0.87	2
Vascular reactivity	True	1.27	1
Blood viscosity	False, but found something about blood pressure	0	0
Prostaglandin	True	0.309	6
Arthritis rheumatoid	True, it appears as "Rheumatoid Arthritis"	0.43	2

Four out of five connecting words discovered in [10] have been found in our system and three of them have high rank (1 or 2) in their corresponding semantic types. Notice that the context we use for discovery is article titles and abstracts whereas [10] used Mesh terms assigned by biomedical experts as potential connecting terms candidates. We also notice that our system found interesting linking terms, such as "Hemodynamic" and "Atherosclerosis**"**, where [10] could not detect.

- **Hemodynamic"** is the connecting word in the chain for "Fish oil" – "Hemodynamic" – "Raynaud Disease". The evidence we found support of this relationship is "fish oil diet can reduce the hemodynamic …" and "Hemodynamic response of the digital artery… in systemic scleroderma and in Raynaud's disease". In short, fish oil can reduce hemodynamic and morphological sequelae of chronic hypoxia, which may have therapeutic significance in treating hemodynamic response of Raynaud Disease.
- **"Atherosclerosis"** is another connecting word for "Fish oil" and "Raynaud Disease". The sentences we found as evidence are "Therefore, fish-oil diets exert effective protective control of progression of atherosclerosis during severe atherogenic stimuli"

and “A new peripheral vasodilator prostaglandin E1 in atherosclerosis of lower limb vessels and Raynaud disease”. In short, fish-oil slows the control of the progression of Atherosclerosis and Atherosclerosis is a symptom of Raynaud disease.

5.2.2. Migraine – Magnesium

As another case study, for two topics Migraine and Magnesium, Gopalakrishnan and Kishlay had found connecting words such as propranolol, adenosine triphosphate, calcium, ergotamine, serotonin, norepinephrine, adenine nucleotides, and epinephrine [10]. Table 3 shows how our system behaves.

Table 5.3. Migraine and Magnesium

Connecting Concept	Find?	Weight	Rank in the Semantic type
propranolol	True	0.41	8
adenosine triphosphate	True	0.52	10
calcium	True	1.50	1
ergotamine	True, the system found “Ergot Alkaloids”	0.34	5
serotonin	True	1.49	1
norepinephrine	True	0.79	6
adenine nucleotides	True	0.32	14
epinephrine	True	0.49	13

Table 3 indicates that the system can find all those eight connecting concepts found in Gopalakrishnan and Kishlay’s system[10]. Six words have high rankings within top ten. The system also found some other linking terms that were not found in the previous study [10], such as “Insulin”.

- "**Insulin**" is the connecting word for the "Migraine" – "Magnesium" relationship query. The evidence is "insulin ... involved in the pathogenesis of migraine" and "Magnesium, a second messenger for insulin". It shows that insulin could be a key factor for answering the migraine and magnesium relationship.

5.2.3. Schizophrenia – Phospholipase A2

Gopalakrishnan and Kishlay also used this query for testing, where they found the following connecting words – chlorpromazine, trifluoperazine, receptors dopamine, prolactin, Choline, arachidonic acid, phenothiazines, and norepinephrine [10]. Table 4 indicates how our system behaves in this case.

Table 5.4. Schizophrenia and Phospholipase A2

Connecting Concept	Find?	Weight	Rank in the Semantic type
chlorpromazine	True	0.22	11
trifluoperazine	True	0.07	32
receptors dopamine	True	0.33	5
prolactin	True	1.17	2
Choline	True	0.12	21
arachidonic acid	True	1.18	1
phenothiazines	True	0.05	56
norepinephrine	True	0.3	11

The result shows that the system once again found all connecting words that were found by Gopalakrishnan and Kishlay's system [10]. Three of them have a very high rank (top five) in their corresponding semantic types. We also notice that our system found interesting linking terms, such as "PGE2**"**, where [10] could not detect.

- **PGE2**” is one of the connecting words that are found by our system but not in [10]. The sentences show as evidence for this relationship are “…increased production of PGE2 in schizophrenia” and “main released by PLA2(Phospholipase A2) was PGE2”. We can see both schizophrenia and Phospholipase A2 affect the production of PGE2.

CHAPTER 6. CONCLUSION & FUTURE WORK

In this paper, I develop a tool which can find hidden connecting words between any two biomedical concepts. It uses Medline database as data source and parses the abstracts and titles of articles from the database. It then processes all texts form abstracts and titles with MetaMap to gather a collection of biomedical concepts and organizes concepts into semantic types provided by NLM. Finally, it provides an interface for querying the data source, built on the technique adapted from Swanson's ABC model. The developed tool can produce fairly accurate results and visualize them in a user-friendly way. Comparing with other approaches [9] [10], this approach can find more unapparent connecting words and those words are further categorized under their belonging semantic types. The interface also extracts all the sentences related to those identified links, with which the users can find out evidence showing how those input concepts are connected by identified intermediate terms.

Currently only one level of intermediate terms are discovered for (i.e., the chains of length 1). In future work, I will expand the chains (links) to multiple levels. This is extremely useful when there is little information about concepts of interest. Additionally, I will combine MetaMaped files together (right now there are about 2000 individual MetaMaped files) so that accessing them can be easier. I will also tweak different MetaMap options to compare their performance.

REFERENCES

[1] Belkin, N. J. (1993). Interaction with texts: Information retrieval as information seeking behavior. Information Retrieval, 93, 55–66. http://doi.org/10.1.1.85.7785

[2] Swason, D. R., "Complementary structures in disjoint science literatures", in Proceedings of the 14th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM Press, Chicago, IL, pp 280-289. (1991)

[3] U.S. National Library of Medicine. (2016). MEDLINE®/PubMed® Resources Guide. Retrieved from https://www.nlm.nih.gov/bsd/pmresources.html

[4] Aronson, a R. (2001). Effective mapping of biomedical text to the UMLS Metathesaurus: the MetaMap program. Proceedings / AMIA ... Annual Symposium. AMIA Symposium, 17–21. http://doi.org/D010001275 [pii]

[5] U.S. National Library of Medicine. (2017). MetaMap - A Tool for Recognizing UMLS Concepts in Text. Retrieved from https://metamap.nlm.nih.gov/

[6] Chapman, W. W., Fiszman, M., Dowling, J. N., Chapman, B. E., & Rindflesch, T. C. (2004). Identifying respiratory findings in emergency department reports for biosurveillance using MetaMap. Studies in Health Technology and Informatics, 107(Pt 1), 487–91. http://doi.org/D040004481 [pii]

[7] Zuccon, G., Holloway, A., Koopman, B., & Nguyen, A. (2013). Identify Disorders in Health Records using Conditional Random Fields and Metamap.

[8] Pratt, W., & Yetisgen-Yildiz, M. (2003). A study of biomedical concept identification: MetaMap vs. people. AMIA ... Annual Symposium Proceedings / AMIA Symposium. AMIA Symposium, 529–33. http://doi.org/D030003464 [pii]

[9] Jin, W., & Srihari, R. (2006). Knowledge Discovery across Documents through Concept Chain Queries. Sixth IEEE International Conference on Data Mining - Workshops (ICDMW'06), 448–452. http://doi.org/10.1109/ICDMW.2006.105

[10] Gopalakrishnan, V. (n.d.). Generating Hypothesis: Using Global and Local Features in Graph to Discover New Knowledge from Medical Literature Knowledge-base Creation.

[11] Wikipedia. (2017). Pipeline (software). Retrieved from https://en.wikipedia.org/wiki/Pipeline_(software)

[12] TFIDF. (2008). Tf-idf :: A Single-Page Tutorial – Information Retrieval and Text Mining Retrieved from http://www.tfidf.com/

APPENDIX A. XMLPARSER.JAVA CODE

```
// First, create a new XMLInputFactory
XMLInputFactory inputFactory = XMLInputFactory.newInstance();
// Setup a new eventReader
InputStream in = new FileInputStream(fileLoc);
XMLEventReader eventReader = inputFactory.createXMLEventReader(in);
// read the XML document
MedlineCite med = null;

int i = 0;
while (eventReader.hasNext()) {
      XMLEvent event = eventReader.nextEvent();

       if (event.isStartElement()) {
          StartElement startElement = event.asStartElement();
         // If we have an item element, we create a new item
         if(startElement.getName().getLocalPart().equals(MEDLINECITATION)){
               med = new MedlineCite();
          }

         if(event.asStartElement().getName().getLocalPart().equals(PMID)) {
               event = eventReader.nextEvent();
               med.setPMID(event.asCharacters().getData());
               continue;

          }
                  ……

       }
       // If we reach the end of an item element, we add it to the list
       if (event.isEndElement()) {
            EndElement endElement = event.asEndElement();
          if(endElement.getName().getLocalPart().equals(MEDLINECITATION)) {
           System.out.println("Processing " + (i + 1) + "/" + "30000" + " MedlineCitation");
          MetaMapProcessor mmProcessor = new MetaMapProcessor(med, host, port);
          mmProcessor.startProcess();
          System.out.println("PostProcess started");
           if (mmProcessor.getTitleResult() != null) {
            PostProcessor posProc = new PostProcessor(med, mmProcessor.getTitleResult(), "T",
semLst);
          posProc.startPostProcess();
                  }
 }
```

APPENDIX B. DOCLISTA.XML

```
<?xml version="1.0" encoding="utf-8"?>
<Documents>
    <Document>Studies of visceral pathology in fowls on fish oil diets.</Document>
    <Document>Three strains of rats were fed a fish oil diet to verify their ability to
    incorporate and convert dietary eicosapentaenoic acid (20:5 omega 3) into trienoic
    prostaglandins</Document>
    <Document>5 mole % in response to the fish oil-supplemented diet.</Document>
    ……

</Documents>
```

APPENDIX C. S2CWA.XML

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<S2CW>
  <SemType name="geoa">
    <Concept weight="1.0">SO</Concept>
    <Concept weight="0.5704939964430451">MDA</Concept>
    <Concept weight="0.40775952588681413">Farm</Concept>
    <Concept weight="0.22725752284865017">farms</Concept>
    <Concept weight="0.22725752284865017">Region</Concept>
    <Concept weight="0.22725752284865017">MX</Concept>
    <Concept weight="0.22725752284865017">Australia</Concept>
    <Concept weight="0.22725752284865017">Greenland</Concept>
    <Concept weight="0.22725752284865017">Community</Concept>
    <Concept weight="0.22725752284865017">states</Concept>
  </SemType>
  <SemType name="popg">
    <Concept weight="1.0">Group</Concept>
    <Concept weight="0.17045684615376022">Males</Concept>
    <Concept weight="0.12307120864928749">female</Concept>
    <Concept weight="0.06950540277181888">cohorts</Concept>
    <Concept weight="0.03873760060949701">White</Concept>
    <Concept weight="0.03873760060949701">Consumer</Concept>
    <Concept weight="0.03873760060949701">SPANISH</Concept>
    <Concept weight="0.03873760060949701">aged</Concept>
    <Concept weight="0.03873760060949701">Eskimo</Concept>
    <Concept weight="0.03873760060949701">Healthy Volunteers</Concept>
    <Concept weight="0.03873760060949701">Senegalese</Concept>
    <Concept weight="0.03873760060949701">Japanese</Concept>
  </SemType>
</S2CW>
```

APPENDIX D. C2SENTSA.XML

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<C2sents>
   <Concept name="Fold">
      <ID>116</ID>
      <ID>184</ID>
      <ID>244</ID>
      <ID>270</ID>
      <ID>296</ID>
      <ID>315</ID>
      <ID>434</ID>
      <ID>435</ID>
      <ID>657</ID>
      <ID>728</ID>
   </Concept>
   <Concept name="Water">
      <ID>583</ID>
   </Concept>
   <Concept name="Time, Prothrombin">
      <ID>586</ID>
   </Concept>
   <Concept name="Platelet adhesion">
      <ID>177</ID>
   </Concept>
</C2sents>
```